A Proposal for Defining Interactions

Perry Alexander
The University of Kansas - ITTC

June 27, 2005

1 Declaration and Definition

1.1 Interactions

Interactions are simply packages that define translator functions and functors. Everything defined in the **translator** and **functor** blocks is implicitly exported.

```
interaction $name(parameters)$ between $D_1$ and $D_2$ [ as $D_3$ ]is
  [ export $exportList$ | all ];
  $localDecls$
begin
  begin translators
    [$translatorDecls$]
  end translators;
  begin functors
    [$functorDecls$]
  end functors
  begin combinators
  end combinators;
    [$combinatorDecls$]
end interaction $name$;
```

D_1 and D_2 are the source domains for models. The optional D_3 value is the common super-type of D_1 and D_2 for forming pullbacks using the $*$ operation. The default value for D_3 is the least common super-type of D_1 and D_2. The super-type is guaranteed to exist due to the nature of the semi-lattice. It is possible that $D_1 = D_2$ in situations where translator functions must lift values from included facets into facets of the same type. Functors may also be written in such interactions, but the identity functor will suffice in most cases.

Parameters must be of kind **design**. Locally defined items must be constant and cannot be exported.

1.2 Translator Functions

```
$name(parameters)$ from $x::T_{src}$ in $D_{src}$ to $T_{dest}$ in $D_{dest}$
  [ is $expression$ ]
  [ where $expression$ ];
```

D_{src} is the source domain and D_{dest} is the destination domain. Both must be either D_1 or D_2 defined in the interaction header. T_{src} and T_{dest} are the source and destination types. They will frequently be the same. x is a parameters for use in the translator definition. *expression* is an expression of type T_{dest} that transforms x into a new value defined in D_{dest}. The value of an translator is a function value. The concrete syntax simply assures appropriate definitions.

The **is** and **where** definitions behave like normal function definitions. Both can be left out to define a variable translator. Why anyone would do this is uncertain, but it keeps the definition consistent with function definition.

1.3 Functor Definitions

```
$name(parameters)$ from $x::D_{src}$ to $D_{dest}$
   [ is $expression$ ]
   [ where $expression$ ];
```

Functor definition is nearly identical to translator definition except the x parameter is a facet of type D_{src} rather than a value defined in that domain. The functor is in all ways a function with syntactic sugar added to the defintion to enhance readability and correct functor definition.

1.4 Combinator Definition

```
$name(parameters)$ from $x::D_{src_1}$ and $y::D_{src_2}$ to $D_{dest}$
   [ is $expression$ ]
   [ where $expression$ ];
```

Combinator definition is nearly identical to functor definition except the x and y parameters represent facets from the two source domains rather than a single source domain. The combinator is in all ways a function with syntactic sugar added to the definition to enhance readability and correct combinator specification.

2 Usage

2.1 Interaction Usage

Interactions are included like packages with the **use** clause:

use *interaction*(*parameters*);

This **use** clause makes translator functions and functors defined in the *interaction* available in the scope of the instance.

2.2 Translator Usage

Translators are applied to facet parameters when one facet instantiates another.

label : *facet*(*p't*,...);

where $facet$ is the instantiated facet name, p is an actual parameter instantiating the associated formal parameter and t is the translator function used to move information from the included facet domain to the including facet domain.

Whether the formal parameter associated with p is an input or output parameter to $facet$, instantiating it with $p't$ asserts that the application of t to p must result in a value compatible with constraints on the formal parameter. This holds whether the parameter is an input or an output.

2.3 Functor Usage

Functor usage is identical to function application:

$functor(p_0,p_1,...,F)$

evaluates to the application of a functor to a facet F. The facet is a required parameter. Other specified parameters, $p_0...p_k$, precede the facet parameter in the argument list. This is done to allow currying to specialize functors.

2.4 Combinator Usage

Combinator usage is identical to function application:

$combinator(p_0,p_1,...,F_1,F_2)$

elaborates to the application of a combinator to facets F_1 and F_2. The facets are required parameters. Other specified parameters, $p_0...p_k$, precede the facets parameters in the parameter list. This is done to allow currying to specialize combinators.

3 Interactions and Use Clauses

It is unweildy to specify interactions elements whenever they are used in a specification. This is particularly true of translator functions that can clutter interfaces and reduce readability. Thus the **use** clause defines default interactions for a given specification. An **interaction** is specified in a **use** clause in the same manner as a package. The **interaction** is named and parametеters specified when required:

use $name(p_0,p_1,...,p_n)$;

The **use** clause specifies identifies specific **interaction** definitions in the same manner as packages using the dote notation to identify where the interaction exists:

use $p_0.p_1...name(p_0,p_1,...,p_n)$;

where p_k are **package** names.

A specific element of an **interaction** can be used by identifying it in the use clause:

use $p_0.p_1...name(p_0,p_1,...,p_n).n$;

where n is the name of a functor, translator or combinator function defined in the **interaction**.

When an **interaction** is used by a package, the **export** clause controls visibility in the same manner as for a **package** use. The only distinction is that for translator functions, the translator domains are used to select a translator when none is explicitly defined. Specifically, if a facet $F_{src} :: D_{src}$ is included in a facet $F_{dest} :: D_{dest}$ and translator functions are not specified for parameters of F_{src}, the domains D_{src} and D_{dest} select the translator function. If a single translator function is visible that translates between from D_{src} to D_{dest}, then that translator is used. If multiple translators are visible, then the user must disambiguate in the specifications.

Situations that require the user to expicitly specify translator functions include:

1. Multiple translators exported from a single interaction.
2. Multiple interactions between the same domains each exporting translators
3. No default translator visible between domains

4 Notes and Issues

- Need to decide if we need universal variables for translators, functors and combinators.
- Do functors manipulate facet values or facet AST representations? My preference is AST representations that denote facet values. Need to make this call before defining functors.

5 Change Log

- Sun May 22 22:08:45 CDT 2005 - Added definition for combinator; moved optional parameters for functors and combinators to the beginning of the parameter list to facilitate currying; suggest **use** clause as a possible default specification capability; minor corrections.
- Mon Jun 27 16:17:33 CDT 2005 - Added syntax for the combinator definition section in the main **interaction** definition block; defined an initial proposal for specifying default translator functions.